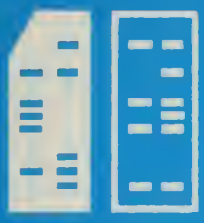
Report No. 439

MACHINE INDEPENDENT COMPILATION OF PL/I: PASS I SYMBOL MANIPULATION

by

Barbara Drahos Mark

June, 1971



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 439

MACHINE INDEPENDENT COMPILATION OF PL/I: PASS I SYMBOL MANIPULATION*

by

Barbara Drahos Mark

June, 1971

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois, 1971.

MACHINE INDEPENDENT COMPILATION OF PL/I: PASS I SYMBOL MANIPULATION

BY

BARBARA DRAHOS MARK
B.S., University of Illinois, 1964

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1971

Urbana, Illinois

ACKNOWLEDGEMENT

I am grateful for the advice of Professor H. George Friedman, who supervised this project. I would also like to thank Mrs. Freda Fischer for her invaluable assistance.

TABLE OF CONTENTS

1. INTRODUCTION

There are many programmers on the University of Illinois campus who use small-memory computers to solve problems and analyze data which is associated with their research. It would be very beneficial to these programmers to be able to use a high level language such as PL/I. However, due to the size of their computers' memories, it has been impossible for them to use such a complex language. Thus a project to remedy this situation was undertaken and this paper is one of the initial results.

The compiler was to be designed to compile PL/I in two passes on the IBM 360 computer and then one or two passes on either the 360 or a specific smaller computer. This project developed into the following specific tasks.

1. Select an initial subset of PL/I for which to write the compiler at present.
2. Select a language in which to write the compiler.
3. Design the first pass to be run on the 360 and implement it.

The design of the intermediate language between the initial 360 passes and the final machine independent passes, and the implementation of these final passes was to be left to future efforts.

1.1 Characteristics of PL/I

PL/I has three important characteristics which make its compiler very large.

1. Modularity - The most inexperienced programmer can use PL/I by simply learning a few I/O statements and important keywords. With this small knowledge, he can easily code programs to solve very complex scientific problems. On the other hand, an experienced programmer might

use his full understanding of the language to write a compiler. People with all ranges of experience can use PL/I. For this reason, the compiler must be able to compile the simplest statements and provide, as default values, all information which is omitted. On the other hand, the compiler must also be able to analyze the most detailed statements and process them.

2. Default - Every variable, every option and every specification has a default value which is assigned by the compiler when some specific value has not been declared by the programmer. Also, identifiers may be used either before or after they have been declared. The compiler must store identifiers, keep account of their individual scopes, and finally assign the proper attributes, declared or default. This requires double storing of identifiers during compilation and continuous checking for previously declared identifiers.

3. Something for Everyone - Unlike the traditional specialized high-level languages, PL/I can be used not only by scientific and commercial programmers but by systems programmers and real-time programmers as well. This requires each PL/I statement to be available for every possible use. For example, the scientific user wants to use the PUT statement to merely print out the result of his computation. However, the commercial user needs to completely edit his PUT statement to properly print business forms.

Eventually, it is important for the compiler to compile most, if not all, PL/I statements, but for the present time, it is most important that the compiler handle those PL/I statements which are important to the scientific programmer. With this in mind, a timetable of PL/I statements

and attributes was set up. This timetable lists, under First Version, those PL/I statements and attributes which are important to the scientific programmer and which are handled by this compiler. At some future time, when the present compiler will be fully working, the statements and attributes under Second Version can be more easily written into the compiler. After the Second Version is working, the Third and Fourth Versions can be implemented and then most PL/I statements and attributes will be included. (See Appendix A.)

1.2 Language Used to Write the Compiler: EOL

When deciding on a language to use to write the compiler, PL/I 360 Assembly language, and EOL were considered. After experimenting with these three languages it was decided that EOL was the best language for this purpose. EOL's suitability to compiler writing and its simplicity make it ideal for writing a compiler for a complex language such as PL/I.

EOL is a stack language. Processing is mostly done using 32 stacks. EOL provides many instructions which permit flexible processing of stacks, such as moving one or more levels from one stack to the beginning or end of a different stack, compressing several levels into one level, separating one level into several different levels, testing whether the top or bottom level of a stack is the same as a given character, etc.

Mass storage is provided by EOL in 2 files. Each file is a list of any number of records and contains exactly one pointer, which may point to any position in the list. EOL provides many instructions which permit one to process files. For example, move the pointer forward until a certain condition is satisfied, store the address of a record directly after the pointer as the top level of an indicated stack, move the

pointer to the beginning of a record whose address is the top level of an indicated stack, etc.

1.3 Design of the Initial Pass

It was decided that Passes I and II should do as much of the initial character manipulation and processing as possible and leave the next pass free to develop the intermediate language between the 360 and the final machine-dependent passes. The initial pass was divided into 3 main routines:

1. Read Routine
2. Statement Recognizer
3. Symbol Dictionary Routine

The Statement Recognizer acts as the coordinator and calls the Read Routine whenever it is ready to recognize a statement. The Read Routine's important function is to return to the Statement Recognizer a single statement, minus statement label prefixes and comments. The Statement Recognizer analyzes the statement and translates it into an intermediate code which it stores in EOL File 2. Whenever the Read Routine or Statement Recognizer recognizes a statement label or identifier, the Dictionary Routine is called. The Dictionary Routine enters the symbol into the Dictionary, part of which is stored in EOL File 1, and part of which is stored in EOL Stacks 0 through 19. The Dictionary is also called at other important events which are listed and described elsewhere in this report.

The design of the Pass I intermediate language and the implementation of the Statement Recognizer is discussed in a separate paper. The final design and implementation of the Read Routine and the Dictionary Routine are described herein.

2. READ ROUTINE

2.1 Functions of Read Routine

1. Starts a new page when 50 lines have been printed, and prints page heading and number.
2. Reads cards from input, prints each card with current statement number, logically deletes first column and last eight columns as per 360 PL/I Compiler convention.
3. Separates statements into "condition-prefixes", "labels", "keyword or first symbol", and "rest of statements".

   Condition-prefixes:

   a. Precede statement labels.

   b. Are contained in parentheses.

   c. Are followed by colon (:).

   Legal labels:

   a. Start with a letter (#, $, @ are considered to be letters).

   b. Contain no characters except those mentioned above and underscore (_).

   c. Can be any length.

   d. Are followed by colon (:). (Blanks may be inserted between label and colon).

   Keywords:

   a. Are the same as labels except are not followed by colon.

   b. Are followed by semi-colon (;), comma (,), left parenthesis (), blank, or equal-sign (=).

   c. Immediately follow the last label or condition-prefix.

Rest:

a. Everything after the keyword up to the semi-colon (;), except comments.

2.2 Stacks Used by Read Routine

Rstwrk:

Used as temporary work stack. Must be empty when Read Routine is called and will be returned empty.

Card:

Must be empty when Read Routine is called. Used to return a single statement to the calling program in the following format:

```
*CØN
(
"cond.prefix.1"
,
"cond.prefix.2"
.
.
.
)
*KEY
"keyword or first symbol"
"symbol.1"
"symbol.2"
"symbol.3"
.
.
.
"symbol.n"
```

"Cond.prefix.1, cond.prefix.2, ..." are any condition prefixes in the statement. Symbol.1 through symbol.n are the "rest of statement". Each statement-label-name, keyword, variable-name, arithmetic-operator, character-string, etc., is entered into a separate level of CARD Stack. The semi-colon at the end of the statement is deleted. When a statement label is recognized by the Read Routine, the following is entered in the top two levels of CARD stack:

*LAB

"statement-label"

Dictionary Routine is called and that routine removes these two levels from CARD Stack, while ignoring anything under these two levels. The colon following the statement-label is deleted.

Number: Used as a counter.

1st level -- statement number counter.

2nd level -- page number counter.

3rd level -- lines printed counter.

When any program prints an error message on the listing page, the number of lines should be added to level three.

Work:

Used to store anything after the semi-colon. At the next call to Read Routine, any data in WORK Stack will be considered to be the beginning of the next statement.

2.3 Example of the Format of Card Stack When Read Routine Returns Control to the Calling Program

Input cards contain:

```
(UNDERFLOW):    LABEL_1:   LABEL_2:   IF  X+Y=3
**VALUE   /*   CØMMENT */  THEN GØTØ
$DØLLARS;      END;
```

When LABEL_1 is read, Read Routine calls DICT (Dictionary Routine) and CARD stack contains:

```
*LAB
LABEL_1
*CØN
(
UNDERFLØW
)
```

When LABEL-2 is read, Read Routine calls DICT and CARD Stack contains:

```
*LAB
LABEL_2
*CØN
(
UNDERFLØW
)
```

When Read Routine is over, CARD Stack contains:

```
*CØN
(
UNDERFLØW
)
*KEY
IF
X
+
Y
=
3
*
*
VALUE
THEN
GØTØ
$DØLLARS
```

The next call to Read Routine will result in no input cards read and CARD Stack will contain:

```
*KEY
END
```

3. DICTIONARY ROUTINE

3.1 Description of the Dictionary at the End of Pass I

This Dictionary contains every variable, statement-label, filename, and entry-name used or declared in the PL/I program. Since PL/I is a highly structured language with a complex system of identifier and statement-label scopes, it is important that the dictionary also keep records of these structures so that reference to identifiers will not be ambiguous. For this reason it is important for Pass I to call the Dictionary Routine every time an identifier or statement-label is being used or declared and also when program structure blocks are being started or ended. In this section the completed dictionary will be described.

There are two types of tables in the Dictionary.

1. Stack tables
2. File tables

1. Stack Tables

A. Format

Every identifier and statement label has a 3-level entry in the stack tables. These three levels are as follows:

level 1: "identifier or statement label"

level 2: "x_0":"pointer"

level 3: "statement number list"

x_0 may take values 0, 1, 2, or 3 as follows:

0 indicates an identifier which is not external.

1 indicates an external identifier.

2 indicates a statement label which is not external.

3 indicates an external statement label.

Pointer may be a file address, a code word, or blank.

File address points to the location in File Table for this identifier.

Code words may be as follows:

A indicates that this identifier has been declared implicitly and has been given default values: Internal, automatic, float, decimal.

I indicates that this identifier has been declared implicitly and has been given default values: Internal, automatic, fixed, binary.

SIF indicates a contextually declared stream input file name.

SOF indicates a contextually declared stream output file name.

L indicates a contextually declared entry statement label.

CSN indicates a contextually declared character string name.

Blank indicates that this identifier is a non-entry statement label and does not need a special file.

Statement-number list contains numbers, separated by commas, of statements where the identifier or statement label has been used. In the case of explicitly or contextually declared identifiers, the first statement number indicates the statement in which the identifier was declared. In the case of statement labels, the first statement number in the list indicates the statement of which this is the label.

B. Arrangement of stack-tables

Each procedure block and begin block has its own stack-table listing identifiers and statement labels whose scope is limited to that block. The statement label of the PRØCEDURE ØPTIØNS(MAIN) statement and all identifiers declared to be external are listed in stack 0. The identifiers

declared in the first inner block, all statement labels in the first inner block, and all undeclared symbols are listed in stack 1. The identifiers in the second inner block and all statement-labels in the second inner block are in stack 2. A maximum of 20 inner blocks can be used.

External identifiers and external statement-labels have 2 or more entries in Stack Table: one entry in Stack 0, for easy reference at the beginning of Pass II, and one entry in every inner block in which it is declared external, to prevent ambiguity.

Example:

Statement-number:	Statement:
1	A: PRØCEDURE ØPTIØNS (MAIN);
2	DECLARE B;
3	C: GØTØ D;
4	D: BEGIN;
5	DECLARE B;
6	DECLARE E;
7	E;
8	DECLARE H EXTERNAL;
9	END;
10	F: BEGIN;
11	DECLARE F;
12	G;
13	END;
14	E;
15	END;

STACK 0	STACK 1	STACK 2	STACK 3
A	B	B	F
3:File address 1	0:File address 3	0:File address 7	0: File address 10
1	2	5	11
H	C	E	
1:File address 2	2:	0:File address 8	
8	3	6,7	
	D	H	
	2:	1:File address 2	
	4,3	8	
	F		
	2:		
	10		
	E		
	0:A		
	14		
	G		
	0:A		
	12		

2. File Tables

Entries in File Table are made only when an identifier is declared explicitly or contextually, as in the case of a statement-label used to label a statement. Once they are entered they are never removed or changed. The format of each file is as follows:

STRUCTURE	TERMINAL	LEVEL	KIDS	PARENT,PARENT,...,*			
CODE-LETTER	*PREC	*SCALE	*ENTRY RAWDTA	*RETURNS RAWDTA	*CHAR LNGTH	*DIM	*INIT

*used as dividers between the last seven items in the file

STRUCTURE: one-character symbol which takes value 0 or 1.

Value 0 indicates a non-structured identifier. In this case the rest of line 1 will be skipped and STRUCTURE will be followed by CODE-LETTER. Value 1 indicates a structured identifier, and STRUCTURE will be followed by TERMINAL.

TERMINAL: one-character symbol which takes value 0 or 1.

Value 0 indicates non-terminal. In this case the file will contain only the top line. Value 1 indicates terminal. In this case the file will contain both lines.

LEVEL: two-character symbol equal to level of identifier in the structure.

KIDS: two-character symbol equal to number of identifiers in level directly below identifier.

PARENT: Name of identifier in level which contains identifier.

Contains 000 if level 1.

CODE-LETTER: one-character symbol which may take following values:

B - Label variable

C - Character

D - Character varying

E - File

F - Entry

G - Returns

J - Decimal

K - Decimal fixed

L - Decimal float

M - Binary

N - Binary fixed

O - Binary float (single)

P - Fixed

Q - Float

PREC: contains the number of digits indicated following BINARY, DECIMAL, FIXED, FLOAT, or REAL attributes.

SCALE: contains the scale factor as given in the precision attribute FIXED.

ENTRY
RAWDTA: contains data following RETURNS attribute which can not be processed at this time.

CHAR
LENGTH: contains declared length of CHARACTER attribute.

DIM: contains value of dimensions for indexed identifier.

INIT: contains values declared for INITIAL attribute. Or contains values declared for LABEL attribute. Or if FILE attribute was declared, INIT contains a 5 character field of binary characters.

Each character represents a file attribute. The character will be 1 when that attribute has been declared, 0 otherwise. The file attributes thus represented are:

INPUT

OUTPUT

STREAM

UPDATE

PRINT

Examples:

```
DECLARE BINDIG BINARY FIXED (5) INITIAL (10101);
```

BINDIG will have file: 0N*5******10101

```
DECLARE FILENAME INPUT;
```

FILENAME will have file: 0E*******10000

```
DECLARE DECNUMB (5,2) DECIMAL (6);
```

DECNUMB will have file: 0J*6******5,2*

```
DECLARE LABELNAME LABEL EXTERNAL (A,B,C);
```

LABELNAME will have file: 0L*******A,B,C

```
PUT FILE (OUTPUT) EDIT (X,Y,Z) (A(10)) PAGE;
```

OUTPUT will have file: 0E*******01100

```
DECLARE 1A,
          2B,
            3C FIXED,
            3D,
              4 E CHAR,
              4 F CHAR,
            3G  FLOAT,
         2H,
           3 I FIXED,
           3 J FLOAT;
```

(It will be understood that a points to the file for A, b points to the file for B, etc.)

a → 100102000

b → 100203a,*

c → 110300B,A,*K*******

d → 100302B,*

e → 110400D,B,A,*C*******

f → 110400D,B,A,*C*******

g → 110300B,A,*L*******

h → 100202A,*

i → 110300H,A,*N*******

j → 110300H,A,*O*******

3.2 List of Events When Dictionary Routine is Called and What it Does at Each Event

In this section the work of the Dictionary Routine will be described. The description of each event will be divided into three sections. 1. Information which the Dictionary Routine expects to find. 2. What the Dictionary Routine will do. 3. What the Dictionary will look like when it returns control to Pass I.

Following is a list of stacks used by Dictionary Routine and a brief description of their purpose.

Stacks 0-19: Used to store declared identifiers from blocks which have been ended.

Stack 0: Also used to store identifiers declared external.

Stack 1: Also used to store undeclared (implicitly declared) identifiers at the end of Pass I.

Stacks 0-5: Used as temporary workspace in analyzing structured identifiers.

SYMLST: Used as temporary workspace in analyzing DECLARE statement.

FILESTK: Used same as SYMLST.

STREAM: Used by Pass I to transfer information to Dictionary.

NUMBER: Contains current statement number.

WORK: Temporary work stack.

WORKSTK: Temporary work stack.

P: Contains numbers of stacks used to store declared identifiers from inner blocks.

PP: Contains numbers of stacks which will be used to store declared identifiers from inner blocks which are not yet ended.

CONTEXT: Work space and temporary storage of contextually declared identifiers.

DECL: Work space and temporary storage of Stack Tables from current unended block.

UNDECL: Work space and temporary storage of undeclared identifiers.

Event 1.

1. *LAB is in top level of STREAM. Statement-label is in second level of STREAM.

2. Dictionary Routine deletes *LAB from STREAM. It then searches for this statement-label in UNDECL and in DECL. If already entered in UNDECL, it concatenates current statement number to front of statement number list and initiates a file in File table and an entry in DECL and deletes entry in UNDECL. If already entered in DECL, it considers this an error and prints an error message. Otherwise, it initiates a file in File table and an entry in DECL. The statement-label is deleted from STREAM and control is returned to Pass I.

3. STREAM is returned with the top two levels deleted. DECL, UNDECL, and File Table contain any information previously unentered concerning the statement-label.

Event 2.

1. BEGIN or PRØCEDURE is in top level of STREAM.

2. This means that a new block is being started. DECL and UNDECL must be empty to process identifiers in the new block. PP is pointing to the stack which is ready to store declared identifiers from the current block, so temporarily store DECL and UNDECL there: all of DECL at the bottom, an asterisk, and all of UNDECL on top. Then increment P by 1 to show a new stack is needed to store declared identifiers from the new block.

Move the number of the new stack to PP to show that it hasn't been used yet and return control to Pass I.

3. STREAM will remain exactly the same. DECL and UNDECL will be empty. P and PP will both contain a new number and one of the storage stacks will be temporarily storing the previous contents of DECL and UNDECL.

Event 3.

1. DECLARE is in first level of STREAM and the rest of the declare statement is contained in subsequent levels.

2. Dictionary Routine analyzes each identifier being declared and initiates a file in File table for each. It then searches UNDECL to see if it has been previously used. If so, it concatenates present statement numbers to front of statement number list, uses this list to initiate an entry in DECL, and deletes entry in UNDECL. If not in UNDECL then Dictionary Routine searches DECL. If found there, an error message is printed that this identifier has been declared twice. The second declaration is deleted. If not in DECL then an entry in DECL is initiated. This process is completed for each identifier contained in the declare statement. Then control is returned to Pass I.

3. STREAM will be empty. Since no other subprogram in Pass I needs a declaration statement, Dictionary Routine deletes it from STREAM as it is being analyzed. DECL contains new entries for identifiers in declare statement. These entries point to the new files in file table.

Event 4.

1. END found in top level of STREAM. A statement label may or may not be in second level of STREAM.

2. If this END is the end of a DO Loop, there will be a *DØ in DECL. If so, delete *DØ and return to Pass I. If not then this END is the END of a PROCEDURE or BEGIN block. PP is pointing to empty stack which is reserved to hold declared identifiers for the block being ended, so Dictionary Routine empties DECL into that stack. Then that number is deleted from PP. END is temporarily deleted from STREAM. If a statement-label is found in STREAM, then the next stack, pointed to by PP, is searched for this statement-label. If there, then END is returned to STREAM and control is returned to Pass I. If not there, then that block is ended also. This means that all the identifiers in the stack (to which PP is pointing) must be compared to identifiers in UNDECL, and statement-number lists concatenated when necessary. Then the temporarily stored undeclared identifiers are returned to UNDECL and the explicitly declared identifiers remain in their permanant storage stack. Now the next number is deleted from PP and the next number stack is searched for the statement lab. This process is continued until the statement label is found, or until PP equals 0. In the first case, END is reentered in STREAM and control is returned to Pass I. However, if PP equals 0 this means that this END is the end of the program. In this case, all entries in UNDECL are assigned A or I (as described in Sec. 3.1.1) and entered in Stack 1. All entries in DECL are moved to Stack 1. Then each Stack file and File file is read and a statement is printed on the PL/I printout concerning each identifier and statement-label. Then END is returned to STREAM and control is returned to Pass I.

3. At the end of an inner block, the declared identifiers and statements labels whose scope is over are stored in their permanant storage table. DECL contains formerly declared identifiers and statement labels whose scope is not yet over. At the end of the program, all identifiers and statement-lables are stored in their permanant storage stacks. STREAM remains the same in both cases.

Event 5.

1. Identifier in top of STREAM.

2. Dictionary Routine searches UNDECL for previous entry of this symbol. If found the current statement number is added to the statement-number list and control is returned to Pass I. If not found, Dictionary Routine searches DECL for previous declaration. If found, current statement number is added to statement number list and control returned to Pass I. If not found, then entry is initiated in UNDECL.

3. STREAM remains the same. New information has been added to appropriate stack table in Dictionary.

Event 6.

1. *EN is on top of STREAM and an identifier is in second level of STREAM. This occurs when the statement recognizer encounters an identifier which by context must be an entry name.

2. DECL is searched for previous declaration of this identifier. If found, the file in File Table is checked to make sure it is an entry name file. If not, an error message is issued. Otherwise, the present statement-number is added to the statement number list and control is returned. If not found in DECL, UNDECL is searched. An entry is initiated in DECL for

this identifier giving it a OF******* file in the File Table (see Sec. 3.1.1.). If the identifier was found in UNDECL, that entry is deleted and the statement number list is added to the end of the present statement number. An "X," is inserted at the beginning of the list to show that this identifier has not been declared explicitly.

3. STREAM is returned with the *END deleted and the identifier on top of the stack. New information has been added to appropriate tables in Dictionary.

Event 7.

1. *SOF (*SIF) is in top level of STREAM and identifier in second level of STREAM. This occurs in a PUT FILE ... (GET FILE ...) statement.

2. DECL is searched for previous declaration of this identifier. If found the file in the File Table is checked to make sure it is a Stream-output-file (Stream-input-file). If not an error message is issued. Otherwise, the present statement number is concatenated to the end of the statement number list and control is returned to Pass I. If not found in DECL, UNDECL is searched and an entry is initiated in DECL for this identifier giving it a OE*******01100 (OE*******01010) file in File Table. (see Sec. 3.1.1.). If the identifier was found in UNDECL, that entry is deleted and the statement number list is concatenated to the end of the present statement number. An "X," is inserted at the beginning of the statement number list to show that this identifier has not been declared explicitly.

3. STREAM is returned with the *SOF (*SIF) deleted and the identifier on top of the stack. New information has been added to appropriate tables in Dictionary.

Event 8.

1. *L in top level of STREAM and an identifier in second level of STREAM. This occurs in a GØTØ ... statement. The *L indicates that the identifier is a label-constant or scalar-label-variable but that this is not the statement for which it is the label.

2. Dictionary Routine searches DECL for a previous declaration of this identifier. If there, it makes sure it has been declared to be a statement label. If not, issue an error message. Otherwise, concatenate statement-number to statement-number list and return to Pass I. If not found in DECL, initiate an entry in DECL with the second level as follows: "2:". Concatenate the present statement number to the end of "X," and if an entry is found in UNDECL, concatenate the statement number list from there to the end. Delete entry in UNDECL.

3. STREAM will have the statement label on the top level. The *L will be deleted. New information has been added to appropriate tables in the Dictionary.

Event 9.

1. DØ in top level of STREAM.

2. "*DØ" is entered in DECL. When the next END is read, indicating the end of the DØ loop, the *DØ will be deleted from DECL.

3. STREAM will remain the same. DECL will be holding *DØ until the end of the DØ Loop.

3.3 Errors Detected

The Dictionary Routine detects the following errors. It is hopeful that they are self-explanatory.

UMIP30I: xxxxx has been given an illegal attribute, yyyyy, which will be deleted.

UMIP31I: Misplaced) in declare statement.. It will be deleted.

UMIP32I: Missing comma in declare statement. A comma will be inserted.

UMIP33I: Misplaced comma in declare statement. It will be deleted.

UMIP34I: xxxxx has been declared more than once in the same block. This declaration will be deleted.

UMIP36I: A structured element in declare statement number xxx has no level number. It is assumed to have the same level number as the previous identifier in the structure.

UMIP37I: xxxxx is not an entry name.

UMIP38I: xxxxx is not a stream output file name.

UMIP39I: xxxxx is not a stream input file name.

UMIP40I: A period comes at the end of a statement. It will be deleted.

UMIP41I: One of the structured identifiers in statement xxx has not been previously declared. It cannot be used.

UMIP42I: The statement label in statement xxx is not the label of any procedure or begin statement. It is assumed that this end statement ends the program.

UMIP43I: xxx is not a statement label.

UMIP44I: xxx is not a character string.

UMIP45I: xxx is not an entry name.

BIBLIOGRAPHY

Engle, John T. "PL/I Declaration Pass: I. The Table Structure," Department of Computer Science, University of Illinois, Urbana, Illinois. Report No. 771, 1968.

"IBM System/360 Operating System PL/I Language Specifications," IBM Systems Reference Library, Form No. C28-6571-4, International Business Machines Corporation, White Plains, New York. 1966.

Lukaszewicz, Leon, and Nievergelt, Jurg. "EOL Report," Department of Computer Science, University of Illinois, Urbana, Illinois. Report No. 241, 1967.

_____, "EOL Programming Examples: A Primer," Department of Computer Science, University of Illinois, Urbana, Illinois. Report No. 242, 1967.

APPENDIX

The following types of PL/I statements and attributes are to be implemented in the proposed PL/I compiler.

First Version

STATEMENTS:

Assignment (scalars, arrays)

BEGIN

CALL (SUBROUTINE)

DECLARE

DISPLAY (REPLY)

DØ

END

EXIT

GET (FILE--SKIP)

GØ TØ (CONSTANT, VARIABLE)

IF

NULL

PROCEDURE (EXTERNAL)

PUT (FILE)

RETURN

STOP

ATTRIBUTES:

AUTOMATIC & STATIC

BINARY, DECIMAL

CHARACTER

DIMENSION

ENTRY

EXTERNAL, INTERNAL

FILE

FIXED, FLOAT

INITIAL

INPUT, OUTPUT, UPDATE

LABEL

LENGTH

OPTIONS

PARAMETER

PRECISION

PRINT

REAL

RETURN

STREAM

VARIABLE

VARYING

Second Version

STATEMENTS:

ASSIGNMENT (STRUCTURE, by NAME)

CLOSE

ENTRY

FORMAT

GET (FILE--COPY)

ON (SYSTEM)

OPEN

PROCEDURE (INTERNAL)

ATTRIBUTES:

BIT

BUILTIN

COMPLEX

LIKE

BIT STREAM

Third Version

STATEMENTS:

DEFAULT

DELETE

LOCATE

ON (SNAP)

READ

RELEASE

REVERT

REWRITE

SIGNAL

UNLOCK

WRITE

ATTRIBUTES:

CONTROLLED

BACKWARDS

BUFFERED UNBUFFERED

CONNECTED

DEFINED

DIRECT

ENVIRONMENT

EVENT

GENERIC

IRREDUCIBLE, REDUCIBLE

KEYED

PICTURE

POSITION

RECORD

Fourth Version

STATEMENTS:

ALLOCATE

BEGIN (ORDER/REORDER)

CALL (TASK, EVENT, PRIORITY)

DELAY

DISPLAY (EVENT)

FETCH.

FREE.

GET (STRING, BIT STRING)

INCORPORATE

PROCEDURE (RECURSIVES, ORDER/REORDER)

PUT (STRING, BET STRING)

WAIT

ATTRIBUTES:

ALIGNED and UNALIGNED

AREA

BASED

EDCLUSIVE

OFFSET, POINTER

SECONDARY

SIZE

TASK


0
1
U.S.A.
2
3
4
5
OREGON RULE CO
1
U.S.A.
2
3
4
5